Multi-threaded File Transfer Utility

Project Description:

The aim of this project is to design and implement a multi-threaded file transfer utility using C programming language. The utility will enable users to transfer files between multiple systems in a network, while utilizing the concepts of threads, processes, system calls, locks, and mutex.

The project will consist of the following components:

- User interface:

  The user interface will allow the user to select files for transfer and to specify the destination system and directory for the transfer.

  Sample instance in command line:

  ```
  Multithreadedfiletransfer [source_path] [destination_address] [file splitting size]
  ```

- File transfer module:

  The file transfer module will be responsible for the actual transfer of files between the source and destination systems. The module will use sockets for communication between the systems and will utilize multi-threading to transfer multiple files simultaneously.

- Process management:

  The project will utilize fork() system calls to create child processes for managing the file transfer operations.

- Locks and mutex:

  The project will utilize locks and mutex to ensure that multiple threads and processes do not access critical sections of code simultaneously.

- Error handling:

The project will have a robust error handling mechanism that will notify the user of any errors that occur during the file transfer operation.

Deliverables:

The project will consist of the following deliverables:

1. Source code: The C source code for the file transfer utility.
2. User manual: A user manual explaining how to use the file transfer utility.
3. Presentation: A presentation describing the project and its implementation.
4. Testing report: A testing report detailing the results of testing the file transfer utility.
5. Project report: A project report detailing the design and implementation of the file transfer utility.

Timeline:

The project will be completed over a period of 8 weeks, with the following timeline:

Week 1: Project proposal and planning

Week 2-3: Implementation of user interface and file transfer module

Week 4-5: Implementation of process management using fork() system calls

Week 6: Implementation of locks and mutex

Week 7: Testing and error handling

Week 8: Finalization of project report, testing report, user manual, and presentation.

For more Information:

- Thread and Process:

  Threads are lightweight processes that enable concurrent execution of tasks within a process. In this project, multi-threading will be used to transfer multiple files simultaneously. Processes, on the other hand, are independent units of execution that can run concurrently with other processes. The project will use fork() system calls to create child processes for managing the file transfer operations.

- Fork():

  The fork() system call is used to create a new process by duplicating the calling process. The new process is called the child process, and the original process is called the parent process. The child process has a new process ID (PID), but it inherits the memory space and file descriptors of the parent process. In this project, fork() will be used to create child processes for managing the file transfer operations.

- System Calls:

  System calls are functions provided by the operating system that enable user programs to interact with the system. In this project, system calls such as socket() will be used to establish communication between the systems for file transfer.

- Locks and Mutex:

  Locks and mutex are used to prevent simultaneous access to shared resources. In this project, locks and mutex will be used to ensure that multiple threads and processes do not access critical sections of code simultaneously, thereby preventing race conditions and data corruption.

To clear up the ambiguity:

In this project, fork() and threads serve different purposes and are used for different tasks.

fork() is used to create a new process by duplicating the calling process. The new process is called the child process, and the original process is called the parent process. In this project, fork() will be used to create child processes for managing the file transfer operations. Each child process will be responsible for transferring a specific file. Since each child process will have its own memory space, it can run concurrently with other child processes and parent process.

Threads, on the other hand, are used for concurrent execution of tasks within a process. In this project, multi-threading will be used to transfer multiple files simultaneously within a child process. Each thread will be responsible for transferring a specific file, and they will run concurrently within the same child process.

Therefore, in summary, fork() is used to create new processes, while threads are used to enable concurrent execution of tasks within a process. In this project, fork() will be used to create child processes for managing the file transfer operations, while threads will be used for concurrent file transfer within each child process.

Good Luck 😊